A Multipath Framework Architecture for Integrated Services

Srinivas Vutukury
Computer Science Department

J.J. Garcia-Luna-Aceves
Computer Engineering Department

Baskin School of Engineering
University of California, Santa Cruz, CA 95064
{vutukury,jj}@cse.ucsc.edu

Abstract— **A major concern with the IETF proposed Integrated Services (Intserv) architecture for providing Quality of Service is that the amount of reservation state it stores in the routers and the RSVP protocol it uses to maintain the consistency of reservation state may not be scalable to high-speed backbone networks. Because of the large number of flows in the backbone network, the refresh messages associated with RSVP's soft-state mechanism, apart from consuming memory, bandwidth and computing power, can experience significant queuing delays and prevent correct functioning of the soft-state mechanism. For the refresh mechanism to scale, therefore, the reservation state size must be bounded so that delays of time-sensitive refresh messages can also be bounded through adequate bandwidth allocation. In [26], we described the Scalable Multipath Aggregated Routing architecture (SMART) in which the reservation state size is a function of number of destinations rather than number of flows in the network. In this paper, we describe a reservation protocol (AGREE) to maintain this reservation state aggregated along the multipaths. The AGREE protocol, like RSVP, uses soft-states, but also ensures that the refresh messages experience bounded queuing delays. The SMART architecture combined with AGREE protocol is significantly more scalable compared to the Intserv/RSVP model.**

I. Introduction

The IETF proposed the Integrated Services architecture (Intserv) for providing deterministic service guarantees required by real-time multimedia applications such as IP telephony, video on demand, and video conferencing [3], [6], [21], [22], [28]. In Intserv and other QoS architectures ([11], [15] to name a few), the network reserves the required link bandwidth for each application, and then uses a fair scheduling algorithm, such as WFQ [4], at the links to ensure each application receives its allotted bandwidth. This approach has several problems related to its scalability. Firstly, routers in Intserv model must remember each flow's reservation and service each flow according to its reservation. As the links in backbone networks have large capacities, routers are expected to carry large number of flows, which, immediately raises the question as to whether the link schedulers will be able to schedule packets in a timely manner [19], [24]. Secondly, a more serious problem is related to maintaining the consistency of reservations in the presence of resource failures and control message loss. If resource reservations state less than the actual reservations, delays cannot be guaranteed. On the other hand, if resource reservations state more than the actual reservations, there is wastage of resources. To implement robust mechanisms for maintaining resource reservations, IETF proposed the RSVP [29] reservation protocol which uses soft-state refreshing to maintain the reservation state. Because of the large number of flows in the backbone, a serious scalability problem can arise if the volume of refresh messages is large enough to cause, due to congestion, delays and loss of refresh messages. Refresh messages are time-sensitive and such delays and loss can easily destabilize the refresh mechanism. To deliver refresh messages in bounded time, the state sizes and refresh message overheads should be predictable so that the necessary resource provisioning can be made for their delivery.

Some schedulers based on a framing strategy can make scheduling decisions in $O(1)$ for a small price in terms of looser delay bounds [13], [23]; link scheduling, it appears, does not seem to present a serious problem even when per-flow scheduling is used. However, solutions for robust reservation maintenance do not seem to be as forthcoming. One may argue that, with current fast processors, high-speed links and inexpensive memory, the high volume of refresh messages associated RSVP can still be processed even when straightforward per-flow processing is used. While this is debatable, the main concern is that the size of the reservation state and refresh message overhead are dictated by the number of flows. When the volume of refresh messages is high, the effect of queuing delays due to congestion can no longer be ignored, even when the refresh messages are forwarded with highest priority. When refresh messages are delayed, flows lose their reservations and potentially destabilize the refresh mechanism.

Several partial solutions, such as aggregation, state compression and message rate control, have been proposed to reduce the volume of RSVP refresh messages [1], [17], [27]. Our approach aims at using reservation state that depends only on the network parameters (number of destinations and classes) rather than arrival patterns of the end user flows. This gives network designers more leverage to bound the bandwidth requirement of refresh messages, and can give them their fair share of the bandwidth by treating them as another real-time flow at the links, for example. Therefore, our goal is to develop an architecture that replaces the per-flow state and per-flow processing with mechanisms whose complexity is essentially determined by the network parameters.

In [26] we proposed a framework architecture which we call SMART (Scalable Multipath Framework Architecture) and which replaces the full reservation state of Intserv with a much smaller state that is bounded and whose bound is determined a priori from the network structure. The SMART architecture achieves this by using a fixed set of paths to each destination and by aggregating flows that share those paths. The core routers maintain state only for aggregated flows and process only aggregated flows. The reservation state is drastically reduced and, more importantly, the state size arising out of these aggregation techniques *is a function of the network parameters rather than the number of user flows*, and thus is easily bounded. This paper focuses on the signaling protocol that manages the aggregated reservation state. We propose the AGREE protocol (AGgregate REservation Establishment protocol) that uses soft states as in RSVP, but refreshes state on a per-aggregate basis

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under grants F30602-97-1-0291 and F19628-96-C-0038.

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **2000**	2. REPORT TYPE	3. DATES COVERED **00-00-2000 to 00-00-2000**
4. TITLE AND SUBTITLE **A Multipath Framework Architecture for Integrated Services**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**5**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

rather than per-flow basis. The protocol can correct inconsistencies in reservations resulting from link failures and from refresh message loses. We believe that the SMART/AGREE model represents a new approach to scalable architectures for providing deterministic guarantees.

The paper is organized as follows. Section II gives a brief overview of the SMART framework architecture. Section III discusses the main topic of this paper, the AGREE reservation protocol. Section IV discusses related work. Section V concludes the paper.

II. Overview of The SMART Architecture

A key idea in the SMART architecture is to use fixed *multipaths* for each destination [26]. A multipath a is directed acyclic sink graph rooted at a destination. Flows of a particular destination are always established along the corresponding routing graphs and are merged with other flows that share the same multipath. Flow classes based on the notion of *burst-ratio* was introduced to facilitate merging of flows, while continuing to provide deterministic guarantees. The end result is that routers only maintain state only on per-class per-destination basis, while offering remarkable tradeoff between complexity and performance. The rest of this section describes SMART to provide necessary context for presenting the AGREE protocol in the next section.

Multipaths: A straightforward approach to achieve per-destination aggregation is to always setup flows along the shortest path (measured in hop-count). Routers then have to maintain only the total rate of traffic they receive for each destination and class and the reservation on the link to the neighbor that is on the shortest path to the destination. The total reservation state that needs to be maintained is reduced to $O(NQ)$, where N is the number of destinations and Q is the number of classes. However, the single-path approach results in high call-blocking rates, i.e., flow requests are rejected even when there are other paths which can satisfy the flow's bandwidth. To improve call-acceptance rates, alternate paths must be used, but using routing state for each alternate path increases the amount of state in the routers. The key to controlling this is to capture alternate paths for each destination succinctly using multipaths. Formally, let S^i_j be the subset of neighbors such that packets received by router i destined for j are only forwarded to neighbors in S^i_j. With respect to a destination j, the successor sets S^i_j of all routers i define a successor graph containing the links $SG_j = \{(m,n) | n \in S^m_j,\ m \in N\}$. Flows of destination j are setup only along SG_j, so that routers only have to store the successor sets S^i_j, which ofcourse is bounded by the number of neighbors. If S^i_j is restricted to one neighbor, the successor graph reduces to a tree as in single-path routing. The goal is to use successor sets as large as possible without creating loops in SG_j, because more richly connected the successor graph SG_j is, the better the call-acceptance rates will be. For the OSPF and IS-IS protocols, $S^i_j = \{k \mid k \in N^i \wedge D^k_j < D^i_j\}$, where D^i_j is the shortest distance of i to j measured in number of hops. A shortcoming of successor graphs provided by OSPF and IS-IS is that they do not use the full connectivity of the network as some links are not used in paths to a destination. To improve on the connectivity, we define *multipath successor graph (MSG)* based on the successor set $S^i_j = \{k \mid k \in N^i \wedge D^k_j < D^i_j \vee (D^k_j = D^i_j \wedge k < j)\}$.

Our intuition for using MSG follows from observing the dynamics of widest-shortest (WS) path selection strategy [14], [16]. The scheme first selects valid paths that are the shortest, and as bandwidth on the shorter paths is consumed, longer paths are tried. The resulting effect is that request are first tried along the MSGs, and even though MSGs put a restriction on the choice of paths that can be used, a majority of flow requests can be satisfied, and we believe the performance lost due to this restriction is an acceptable tradeoff for the resulting scalability.

A key benefit of establishing flows along the MSG is that the routing state is of $O(N)$, where N is the number of destinations in the network; there is one successor graph for each destination. All flows with a particular destination j are aggregated and they collectively share the bandwidth allocated on MSG for j. This bounded state can be maintained more easily than per-flow state. Another advantage is that flows can be established on a hop-by-hop basis by choosing the next router from the successor set at each hop. When there is more than one successor to choose from, the next-hop can be chosen using a load-balancing heuristic such as widest link. We explore the performance of such schemes in other publications. The performance can be further improved over MSGs if we define the enhanced successor set S^i_j as $S^i_j = \{k | k \in N^i \wedge D^k_j \leq D^i_j\}$. That is, the successor set with respect to a destination consists of all neighbors of a router that are closer to or at the same distance from the destination. Note that each router, unlike in MSG, includes the "peer" neighbor in its successor set, which can cause packets to loop endlessly. To prevent this, special mechanisms for forwarding packets must be used. We explore those in a future publication. A drawback of multipaths is that packets may arrive out of order at the destination, but this is not a concern because end-to-end delay bounds are provided which can be used by the application to detect loss of data; real-time applications use a playback time and packets need to only arrive within that time frame, and order does not matter.

Flow classes: As stated earlier, flows that share a multipath are merged and handled collectively. Because merging of flows removes isolation between flows, merging should be done systematically without effecting end-to-end delays. In our prior work [26], we introduced the notion of burst-ratio and described a technique for defining flow classes. The flow classes are such that if two flows belonging to the same class g are merged, then the resulting flows also belongs to the same class g. Similarly, if a flow is divided into two or more flows in fixed proportions along multiple successors, each of the flow belongs to the same class g. We assume that there are Q flow classes defined. A class identifier g is included in every packet that belongs to a flow of class g. The class identifier is used by the schedulers at the links to perform class-based fair-scheduling where all packets belonging to the same class are treated as same flow. So the link scheduler processes at most Q queues. Because of the non-zero size of packets, the link schedulers introduce burstiness which is, however, removed at the receiving end by using per-aggregate token-bucket shapers rather than per-flow shaping as in [12]. Since this paper focuses mainly on the reservation maintenance protocol AGREE, we assume a fluid model and address per-hop per-aggregate shaping in another publication.

Routing table structures: As there are more than one next-hop at a router for packet forwarding, the bandwidth for each of the next-hops must be specified. For each $k \in S^i_j$, let $B^i_{j,g,k}$ specify the bandwidth for traffic of class g and destination j received by the router i that is forwarded to neighbor k, and for $k \notin S^i_j$, let

$B^i_{j,g,k}$ specify the rate of traffic that is received from the neighbor k. Define the set $B^i_{j,g} = \{B^i_{j,g,k} | k \in S^i_j\}$. The routing table entry for j and class g is of the form $\langle j, g, B^i_{j,g}, S^i_j \rangle$. Assuming the network does not lose packets, we have $\sum_{k \in S^i_j} B^i_{j,g,k} = \sum_{k \notin S^i_j} B^i_{j,g,k} + I^i_{j,g}$, where $I^i_{j,g}$ is the traffic of class g entering at i for destination j.

The packet forwarder at i uses a weighted round robin discipline to allocate packets to next-hops in proportion to their bandwidths. When it receives a packet for router j, it determines the next-hop k for this packet using $B^i_{j,g}$, and puts the packet in the queue g of link (i, k). The time complexity of weighted round robin discipline is constant because there are fixed number of neighbors.

III. The AGREE Protocol

The goal of the AGREE protocol is to maintain the consistency of reservations. If at each router i for all destinations j and classes g, $\sum_{k \in S^i_j} B^i_{j,g,k} = \sum_{k \notin S^i_j} B^i_{j,g,k} + I^i_{j,g}$, then reservations are said to be in consistent state. The pseudocode of AGREE is shown in Fig. (1). The AGREE protocol uses soft-states like RSVP and YESSIR, but because reservation state is on per-destination per-class basis, *its reservation refresh messages are on per-destination per-class basis.* Every T_R seconds (refresh time period), for each destination j and g, the router i invokes $AgreeEvent(TIMEOUT, j, g, -, -)$ (the parameters b and k are ignored) for comparing the cumulative reservations of the incoming refresh messages with the current reservations and sending its own refresh messages. A refresh message specifies the destination j, class g and the associated bandwidth b. The source node of each flow sends its refresh messages to the ingress node every T_R seconds, stating its destination, class and its rate. At the ingress node all refresh messages of a particular destination and class are aggregated and a single refresh message is sent to the next-hop. When a flow terminates, the source stops sending its refresh messages and the bandwidth reserved for the flow is eventually timed out and released in the network. The core refresh cycle is shown on lines 02-13. Let the reserved bandwidth on the outgoing links in S^i_j for class g add up to bw. Let the refresh messages received by router i for destination j from neighbors not in S^i_j and refresh messages originating at the router, during the previous refresh period add up to a total bandwidth of bt. Note that the refresh messages originating at the router itself add up to $I^i_{j,g}$. First, bt is compared with bw and if $bt = bw$, the reservations are in consistent state and no bandwidth need to be released. The router simply sends a refresh message to each next-hop $k \in S^i_j$ with current allocated bandwidth $B^i_{j,g,k}$.

Reservations can become inconsistent, i.e., $bt \neq bw$, because of flow terminations, link failures and control message losses. To correct the inconsistencies we consider the two separate cases: (1) $bt < bw$, and (2) $bt > bw$. The first case is handled by lines 6-9 and 12 of the pseudocode. The total incoming bandwidth bt is first divided into b_1,..,$b_{S^i_j}$ such that for each $k \in S^i_j$, $b_k \leq B^i_{j,g,k}$, and then for each $k \in S^i_j$, $B^i_{j,g,k}$ is updated with b_k and a refresh message is sent to k with the new bandwidth b_k. The second case, i.e., $bt > bw$ is generally more difficult, and requires forcing the upstream routers to reduce their outgoing bandwidth. We describe two techniques to correct this inconsistency. The first method uses the fact that the underlying routing protocol (such as OSPF) informs

```
01  PROCEDURE $AgreeEvent$ ($type$, $j$, $g$, $k$, $b$)
02  if ($type = REFRESH$), $B^i_{j,g,k} \leftarrow B^i_{j,g,k} + b$;
03  if ($type = TIMEOUT$), then {
04      $bt \leftarrow \sum_{k \notin S^i_j} B^i_{j,g,k} + I^i_{j,g}$;
05      $bw \leftarrow \sum_{k \in S^i_j} B^i_{j,g,k}$;
06      if ($bt < bw$), then {
07          Divide $bt$ into $b_k$ such that $\sum b_k = bt$ and $b_k \leq B^i_{j,g,k}$;
08          $B^i_{j,g,k} \leftarrow b_k$;
09      }
10      if $bt > bw$ and $state^i_{j,g} = PASSIVE$, then
11          CALL $DiffComp$ ($j$, $g$, $bt - bw$);
12      for each $k \in S^i_j$, send [REFRESH, $j$, $g$, $k$, $B^i_{j,k}$];
13  }
14  if ($type = RELEASE$), then {
15          $B^i_{j,g,k} \leftarrow B^i_{j,g,k} - b$;
16          if ($state^i_{j,g} = PASSIVE$), then
17              CALL $DiffComp$ ($j$, $g$, $b$);
18          otherwise send [ACK, $j$, $g$] to $k$;
19      }
20  if ($type = ACK$ and last ACK message for $j$ and $g$) {
21          $state^i_{j,g} \leftarrow PASSIVE$;
22          send [ACK, $j$, $g$] to $s$, if $s$ is waiting for ACK;
23      }
24  if ($type = SETUP$) then // $s$ is the successor on the path
25      $B^i_{j,g,k} \leftarrow B^i_{j,g,k} + b$; $B^i_{j,g,s} \leftarrow B^i_{j,g,s} + b$;
26  if ($type = TERMINATE$) then
27      $B^i_{j,g,k} \leftarrow B^i_{j,g,k} - b$; $B^i_{j,g,s} \leftarrow B^i_{j,g,s} - b$;

28  PROCEDURE $DiffComp$ ($j$, $g$, $b$) at node $i$
29      if ($b \leq I^i_{j,g}$), then terminate flows for $j$ and
30          class $g$ that add up to at least $b$ and return;
31      $br \leftarrow b - I^i_{j,g}$;
32      Divide $br$ into $b_k$ and $k \notin S^i_j$ such that
33          $\sum b_k = br$ and $b_k \leq B^i_{j,g,k}$;
34      for each $k \notin S^i_j$, send [RELEASE, j, g, $b_k$] to $k$;
35      $state^i_{j,g} \leftarrow ACTIVE$;
```

Fig. 1. Event Handling in the AGREE protocol

all routers about link failures. When a router learns about a failed link, it terminate all flows that use that link. The soft-state refresh mechanism will then eventually release the bandwidth reserved for these flows using the same process outlined to handle case (1). A router only need to remember the path of each flow that originates from it.

The second method uses diffusing computation [10] to correct the inconsistencies. When the router i detects failure of adjacent link (i, k) it invokes $AgreeEvent(RELEASE, j, g, k, B^i_{j,g,k})$ for each j and g. The router updates $B^i_{j,g,k}$ (line 15) and invokes $DiffComp$ if it is in PASSIVE state. The $DiffComp$ procedure first terminates as many flows as possible at the router, and if there is still some bandwidth that must be released to restore consistency, the router distributes the excess bandwidth among upstream neighbors and requests them, using RELEASE messages, to reduce sending required traffic to this router (lines 31-34). The router then enters ACTIVE state indicating that it is waiting for the upstream nodes to reply with ACK messages. When an upstream router receives a RELEASE it repeats the same process. When a router is in ACTIVE state, if it receives RELEASE messages from successor nodes, it immediately sends back an ACK message (line 18). After all ACK messages are received, it transits to PASSIVE state (line 21) and if the transition to ACTIVE state was triggered by a RELEASE message from the downstream message, it sends the ACK message to the successor node that triggered the transition to ACTIVE state (line 22). When flow-setup and terminate messages are received, they are simply forwarded to the next hop after the reservations are modified.

During routing-table convergence, stray release messages may arrive from current upstream nodes. These are safely ignored by immediately sending ACK messages even when the router is in PASSIVE state. Similarly, the refresh messages received from downstream nodes and duplicate refresh messages are ignored. When a neighbor k is added or removed from a successor set, the corresponding $B^i_{j,g,k}$ are reset for each j and g. Note that though we did not explicitly state in the pseudocode, before initiating the diffusing computation, an attempt can be made to reserve the required bandwidth through a new request and only when the request fails the diffusing computation can be triggered.

The AGREE protocol can be said to work correctly if after a sequence of link failures and refresh message losses, and if no new flows are setup and terminated, within a finite time all reservations reflect a consistent state. For correct functioning of the protocol, we assume messages on a link are received and processed in order. This prevents race conditions between flow setup, terminate, refresh and release messages. Because within a finite time the topology stabilizes and the routing protocol ensures that loop-free shortest paths are established for each destination, all diffusing computations terminate and all routers return to PASSIVE state for each class-destination pair. In the AGREE protocol, the release messages and the refresh messages only decrease the reserved bandwidths. Because bandwidths cannot decrease forever, after a finite time no new diffusing computations will be initiated. At this time, at all the nodes the bandwidth specified by refresh messages for a particular bandwidth can only be less than or equal to the reserved bandwidth at that node, otherwise this will again trigger another diffusing computation. If on the other hand refresh messages specify lower bandwidth than reserved bandwidth, then that extra bandwidth is eventually released by the usual timeout process of case (1). So, eventually all reservations must converge to a consistent state.

In each refresh period at most $O(QN)$ refresh messages are sent on a link irrespective of number of flows in the network. Since the bandwidth requirements for refresh messages is known a priori, they can be serviced through a separate queue in the link scheduler and guarantee a delay bound. So refresh messages are never lost due to queuing delays. This is not possible in per-flow architectures as the number of flow on a link cannot be determined a priori. In AGREE, they can only be lost due to link failures, which in backbone network is relatively rare. Even then the AGREE protocol is more resilient to refresh message loss compared to a per-flow architecture. In per-flow architectures *a lost refresh message cannot be distinguished from flow termination* and the router interprets a lost refresh message as a flow termination and attempts to release bandwidth from downstream links. In the following cycle when the refresh message is received correctly, it tries to recover the released bandwidth. In contrast, in AGREE a link can simply *use a null refresh message when it does not carry any traffic for a particular destination and class.* This enables distinguishing flow termination from refresh message loss. When a periodic refresh message is lost, the receiving node recognizes it and continues to use the contents of the refresh message of the previous cycle. In the following cycle, if a refresh message is received correctly, the new refresh message is used. In essence, *refresh messages are sent irrespective of the presence of flows in a synchronous manner* which is only possible because AGREE's reservation state is based on network parameters. This model is scalable, because the worst case bounds on state size depend on the number of active destinations and classes rather than the number of individual flows.

IV. Related Work

The scalability problem of the RSVP protocol is well-known and there have been several proposals to reduce its refresh message overhead [1], [17], [18], [27]. The technique in [27] applies a compression algorithm (CRC-32 or MD5) on the reservation state to produce a digest and refresh messages are exchanged for digests rather than for individual sessions. As a result the number of refresh messages is proportional to number of neighbors rather than the number of sessions. Though the number of messages is significantly reduced, new complexities are introduced in the refresh mechanism. The computation of digests requires grouping of sessions according to next-hops and is expensive when sessions are short lived and individual sessions change paths. Also, neighbors may end up in inconsistent state when the messages do not represent the digests and it is not easy to recover from it. It does not fundamentally reduce the amount of state information that needs to be maintained, and refresh message loss cannot be distinguished from flow termination. Our method differs in that the state is significantly reduced through aggregation of flows, so much so that it depends on network parameters and it is feasible to send per-class per-destination refresh message even when there is no corresponding flow on the link, and help distinguish flow termination from refresh message loss. Moreover, these compression techniques, if desired, can also be incorporated into our approach to further enhance it. In [1], several refresh messages are bundled into a single message. This technique provides scaling benefits, but it compromises on error recovery properties. To recover from corrupted internal state, standard refresh messages must be sent in addition to bundled messages, adding complexity to the protocol and its configuration. However, like the previous technique, this technique does not fundamentally reduce the state size and if needed can be incorporated into our AGREE protocol.

Another approach to reducing refresh message volume is to control the refresh interval [17]. This technique also does not fundamentally decrease the state size, and its focus is on reducing the refresh messages. This technique can be used orthogonally to our technique. The YESSIR protocol [18] uses a sender-initiated approach and avoids separation of reservation and path-finding messages. As a result the processing and protocol complexity is reduced. The AGREE protocol is sender-initiated and has similar benefits, but differs in that it manages reservations that depend on network parameters and is closely tied to the proposed multipath framework. Lastly, the multipath reservation maintenance feature of the AGREE protocol can be incorporated into YESSIR and RSVP as extensions or as a separate protocol and is a subject of future work.

The Diffserv architecture [5], [9], proposed to address the scalability of the Intserv architecture, uses no per-flow state in the core routers, but the approach is mainly targeted at providing statistical guarantees and not deterministic guarantees. Approaches similar to Diffserv have been proposed for providing deterministic guarantees [25], [30]. In the SCORE architecture [25], to provide deterministic guarantees without per-flow state management, the per-flow reservation state is carried in the packets of the flows and not stored and maintained in the core. The reservations stated in the packets is then used by the core routers to estimate the aggregate reservation on the links. There are no explicit refresh messages and thus the problems associated with lost or delayed refresh messages do not arise. However, the reservation estimation is dependent on the individual flow behavior and is often inaccurate. It is possible that this kind of estimation based on user-level flows can cause instability and, therefore, it must be decoupled from individual flow behavior. In addition, this approach does not particularly reduce the processing or bandwidth overheads. In [30], a central system, called the bandwidth broker, makes reservations for all the flows in the network, and hence, there is no need for soft-state refresh mechanism and again problems related to lost or delayed refresh messages do not arise. The obvious drawback is that it does not scale to large networks because of its centralized architecture.

Our reservation aggregation schemes differ from those proposed to date. In the aggregation techniques proposed in [2], [8], computing delay bounds in a dynamic environment are not discussed. In BGRP protocol [20], the flows are setup and aggregated along a sink tree for each domain network. The reservation state aggregation in BGRP has some similarities with our approach, but BGRP is targeted at providing differential services in an inter-domain environment. Our approach provides deterministic guarantees in an intra-domain network or a VPN, and through multipaths provides richer connectivity than a sink tree. The aggregation technique proposed for RSVP [7] are meant for aggregating reservation state of flows within a single multicast group. Our schemes aggregate state of flows belonging to different multicast group and as such is orthogonal to aggregation within RSVP.

V. Conclusions

We presented the AGREE protocol for managing reservations established along multipaths. The SMART architecture [26] combined with the AGREE protocol addresses some of the drawbacks of the Intserv/RSVP architecture. The loss and queuing delays that the refresh messages of RSVP experience pose a much more serious threat to the scalability of the Intserv/RSVP model than the storage, processing and bandwidth requirements. Because refresh messages are very time sensitive, unpredictability in their delivery will eventually destabilizes the soft-state refresh mechanism. Our approach to this problem is to contain the volume of refresh messages, and statically determine a bound on refresh message overhead based solely on the network size and independent of the number of end-user flows. Since the message overheads are known a priori, by provisioning network bandwidth and other resources, the refresh messages can be delivered reliably and without delays. The SMART/AGREE is the first architecture framework that addresses time-bound delivery of refresh messages. The key to containing the volume of refresh messages is flow aggregation, and we proposed a multipath flow aggregation which eliminates the need for per-flow state maintenance in the routers and is far more scalable than the Intserv/RSVP model.

The main drawback of SMART/AGREE architecture is that the delay bounds can be loose compared to the tight delays that can be achieved using per-flow processing. Also, the call-blocking rates tend to be higher due to restrictions imposed by multipaths on call establishment. However, we believe these drawbacks are not detrimental and are a reasonable tradeoff for achieving scalability.

References

[1] L. Berger and et al. RSVP Refresh Overhead Reduction Extensions. *Internet Draft, draft-ietf-rsvp-refresh-reduct-05.txt*, June 2000.

[2] S. Berson and S. Vincent. Aggregation of internet integrated services state. *IWQOS*, 1998.

[3] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: An overview. *RFC 1633*, June 1994.

[4] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. *Proc. of ACM SIGCOMM*, pages 1–12, 1989.

[5] D. Black et al. An Achitecture for Differentiated Services. *Internet Draft*, May 1998.

[6] E. Crawley et al. A framework for qos-based routing in the internet. *Internet Draft*, April 1998.

[7] R. Guerin et al. Aggregating RSVP-based QOS Requests. *Internet Draft*, Nov. 1997.

[8] Rampal et al. Flow grouping for reducing reservation requirements for guaranteed delay service. *Internet Draft:draft-rampal-flow-delay-service-01.txt*, July 1997.

[9] Y. Bernet et al. An Framework for Differentiated Services. *Internet Draft*, May 1998.

[10] E.W.Dijkstra and C.S.Scholten. Termination Detection for Diffusing Computations. *Information Processing Letters*, 11:1–4, August 1980.

[11] D. Ferrari, A. Benerjea, and H. Zhang. Network support for multimedia - a discussion of tenet approach. *Computer Networks and ISDN*, 26:1267–1280, 1994.

[12] L. Georgiadis, R. Guerin, V. Peris, and K. Sivaranjan. Efficient Network QoS Provisioning Based on per Node Traffic Shaping. *IEEE/ACM Trans. Networking*, pages 482–501, August 1996.

[13] S.J. Golestani. A Framing Strategy for Congestion Management. *IEEE Journal on Selected Areas in Communications*, pages 1064–1077, Sept. 1991.

[14] R. Guerin, A. Orda, and D. Williams. QOS Routing Mechanisms and OSPF Extensions. *Internet Draft*, April 1998.

[15] S. Kweon and K. Shin. Providing Deterministic Delay Guarantees in ATM Networks. *IEEE/ACM Trans. Networking*, 6(6):838–850, December 1998.

[16] Q. Ma and P. Steenkiste. On path selection for traffic with bandwidth guarantees. *Proc. International Conference on Network Protocols*, October 1997.

[17] P. Pan and H. Schulzrinne. Staged Refresh Timers for RSVP. *Globecom*, Nov. 1997.

[18] P. Pan and H. Schulzrinne. YESSIR: A Simple Reservation Mechanism for the Internet. *Computer Communications Review*, 29(2), April 1999.

[19] A.K. Parekh and R.G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Trans. Networking*, 1:344–357, June 1993.

[20] P.Pan and et al. BGRP: A Framework for Scalable Resource Reservation. *Internet Draft, draft-pan-bgrp-framework-00.txt*, July 2000.

[21] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service: An overview. *RFC 2212*, Sept. 1997.

[22] S. Shenker and J. Wroclawski. General characterization parameters for integrated service network elements. *RFC 2215*, Sept. 1997.

[23] M. Shreedhar and G. Varghese. Efficient Fair Queueing using Deficit Round Robin. *Proc. of ACM SIGCOMM*, 1995.

[24] D. Stiliadis and A. Varma. Rate-Proportional Services: A Design Methodology for Fair Queuing Algorithms. *IEEE/ACM Trans. Networking*, April 1998.

[25] I. Stoica and H. Zhang. Providing Guaranteed Services Without Per Flow Management. *Proc. of ACM SIGCOMM*, Sept. 1999.

[26] S. Vutukury and J.J. Garcia-Luna-Aceves. A Scalable Architecture for Providing Deterministic Guarantees. *Proc. of ICCCN*, Oct. 1999.

[27] L. Wang and et al. RSVP Refresh Overhead Reduction by State Compression. *Internet Draft, draft-wang-rsvp-state-compression-03.txt*, March 2000.

[28] J. Wroclawski. Specification of the controlled-load network element service. *RFC 2211*, Sept. 1997.

[29] L. Zhang and et al. RSVP: A New Resource Reservation Protocol. *IEEE Communications Magazine*, 31(9):8–18, 1993.

[30] Z. Zhang and et al. Decoupling QoS Control from Core Routers: A Novel Bandwidth Broker Architecture for Scalable Support of Guaranteed Services. *Proc. of ACM SIGCOMM*, pages 71–83, 2000.